Example 3 continued. Fitting a mixed effect survival model

A. Onofri, H-P. Piepho and M. Kozak

Contents

The model definition (in JAGS code)

In the previous session (here) we have fitted a survival model to a dataset relating to potato starch grains assessed in size categories. Our analyses assumed that starch grains were independent, an artificial assumption - because they actually were *not* independent, being grouped by photo. Starch grains observed in the same photo can be correlated and thus similar to each other. In our paper, we discuss that neglecting this correlation we incorrectly treated the 2,441 starch grains as 2,441 independent pieces of information, leading to overly small standard errors of our estimates.

Including random effects in a survival model is difficult, unless we use Bayesian methods. In our paper, we already discussed the differences between traditional and Bayesian statistics; therefore, we will not make this point again here, but instead we will concentrate on the fitting process. The model we are going to fit is as follows:

$$y_{ijk} = \mu_j + \gamma_k + \varepsilon_{ijk}$$

where y_{ijk} is the diameter of the i-th grain in the k-th phot for the j-th producer, μ_j is the mean diameter for the j-th producer, γ_k is the random error for the k-th photo, and ε_{ijk} is the residual error term. We will assume that ε is homoscedastic and normally distributed, with mean equal to zero and standard deviation equal to σ. We will also assume that γ (the random photo effect) is normally distributed, independent of ϵ, with mean equal to 0 and standard deviation equal to σ_p. This standard deviation represents the photo-to-photo variability, while the residual standard deviation represents the within-group (within-photo) variability.

To fit the above model, we used the MCMC sampler provided with the freeware software JAGS (Just Another Gibbs Sampler), which can be invoked from an R session with the `rjags` package (Plummer 2016).

First, we need to specify a model (in JAGS code) in a text string (`modelSpec`), like below.

```
# Save BUGS description of the model as a string
modelSpec <- "
data{
for (i in 1:N) {
  zeros[i] <- 0
}}

model{
```

```
for (i in 1:N) {
  exp[i] <- mu[Group[i]] + gamma[Photo[i]]
}

for (i in 1:N1) {
  #Likelihood for left-censored
  S2[i] <- pnorm(high[i], exp[i], tau.e)
  L[i] <- S2[i]     #(Equation 3)
  phi[i] <- -(log(L[i]))
  zeros[i] ~ dpois(phi[i])
}

for (i in (N1+1):N2) {
  #Likelihood for interval-censored
  S[i] <- pnorm(low[i], exp[i], tau.e)
  S2[i] <- pnorm(high[i], exp[i], tau.e)
  L[i] <- S2[i] - S[i]  #(Equation 4)
  phi[i] <- -(log(L[i]))
  zeros[i] ~ dpois(phi[i])
}

for (i in (N2+1):N) {
  #Likelihood for right-censored
  S[i] <- pnorm(low[i], exp[i], tau.e)
  L[i] <- 1 - S[i]  #(Equation 5)
  phi[i] <- -(log(L[i]))
  zeros[i] ~ dpois(phi[i])
}

#Priors
  sigma.e ~ dunif(0, 100)
  sigma.P ~ dunif(0, 100)
for(i in 1:2){
  mu[i] ~ dnorm(0, 0.000001)
} for(i in 1:24){
   gamma[i] ~ dnorm(0, tau.P)
}

#Derived quantities
  sigma2p <- sigma.P*sigma.P
  sigma2e <- sigma.e*sigma.e
  tau.P <- 1 / sigma2p
  tau.e <- 1 / sigma2e
  diff <- mu[1] - mu[2]
}
"
```

This definition contains an (initial) data step, where we create a new variable (`zeros[i]`) by assigning a value of 0 to all observations. We will clarify why this data step is necessary.

Then, a model step follows, where we code the model we want to fit. This returns the expected value for each observation (`exp[i]`). Note the coding with double square brackets (`mu[Group[i]]`), which is used to mean that `mu` takes different values, depending on `Group[i]`.

The next part codes the log-likelihood for the i-th observation. The observations are assumed to be sorted,

so that right-censored observations take the positions from 1 to N1, interval-censored observations take the positions from N1 to N2, and left-censored observations take the postions from N2 to the end of the dataset. For each group, the negative log-likelihood is coded according to the Equations from 3 to 5 in our paper and is stored in the variable `phi`.

At this stage, we need to make JAGS use `phi` to calculate the likelihood for each observation. Such calculation might seem easy, since the likelihood is equal to `exp(-phi)`. Unfortunately, JAGS/BUGS can only calculate the likelihood using a few pre-defined functions, none of which providing a simple direct solution.

This problem can be solved by using the so-called 'zeros trick' (Spiegelhalter et al. 2003): the trick is to set the observed value for the i-th observation as `zeros[i] = 0`, which we have done in the first (data) step. We now need to specify that the observed value (0 for all observations) is distributed according to a Poisson probability function (i.e., one of the available pre-defined functions) with mean equal to phi; we can do this with the command `zeros[i] ~ dpois(phi[i])`. Note that the probability of obtaining zero from a Poisson distribution with mean equal to phi is exactly e^{-phi}. sO, with this "zeros trick" the software can assign the correct likelihood to each individual observation.

Finally, we have to define the priors for all the parameters. For both variance parameters, we selected a uniform distribution from 0 to 100 (`sigma.e ~ dunif(0, 100)`). For μ_1 and μ_2 our prior expectation was that they were normally distributed with a mean of 0 and a very high standard deviation ($\sigma = 10^3$). So high an SD value used as a prior information means that, without looking at the experimental data, we had no idea about the values of these unknown parameters. The priors for μ_1 and μ_2 were set with the command `mu[i] ~ dnorm(0, 0.000001)`, which specifies the normal distribution using the mean of 0 and the precision of 0.000001, the precision being $1/\sigma^2$. Such a parameterisation is used by JAGS and other MCMC samplers, such as WinBUGS.

In the above model definition, we also added some derived quantities, to be calculated from the estimated parameters, such as the difference between μ_1 and μ_2.

Model fitting in R

So, we coded the model and assigned it to a text string (`modelSpec`). Before we start fitting the model, however, we have to

1. Store the model definition in an external text file ('censoredMixedModel.txt'), using the function `writeLines()`.
2. Load the dataset. We will reuse the dataset in the ungrouped form, which we used to fit the survival model. Let's recall that this data set is available in the package `agriCensData` as `starchGrainU`.
3. Sort the dataset in an ascending class order, so that the individuals in the first diameter class (left-censored) begin the data set (from row 1 to row 639) and the individuals in the 5th diameter class (right-censored) end it (from row 2,131 to row 2,441).
4. Define a few variables that are used in the model definition, such as `N` (the number of observations), `N1` (the number of left-censored observations, i.e., 639), and `N2` (the number of left-censored plus interval-censored observations, i.e., 2130).

```
library(agriCensData)
writeLines(modelSpec, con = "censoredMixedModel.txt")
data(starchGrainU)
dataset_jags <- starchGrainU[order(starchGrainU$Class), ]
N1 <- 639
N2 <- 2130
N <- 2441
```

Now we are ready to fit the model. To this aim, we will

1. load the `rjags` library;
2. create two lists: a list of all the data needed for the analysis (`win.data`) and a list of the initial values for the parameters to be estimated (`init`); and
3. send the model specification and the other data to JAGS, using the function `jags.model()` from the `rjags` package; in a few seconds, this function returns samples from the posterior distribution for all the estimated parameters (`res3`).

Once the samples from the posterior have been obtained, we specify that 1000 samples should be discarded as `burn.in`. These samples might have been produced before reaching the convergence, so they might not come from the correct posterior distribution. In other words, we get rid of them.

From the posterior, we obtain the mean and median as measures of central tendency, the standard deviation as a measure of uncertainty, and credible intervals, which are the Bayesian analog to confidence intervals.

```
library(rjags)
win.data <- list(low = dataset_jags$sizeLow,
                 high = dataset_jags$sizeUp,
                 N1 = N1, N2 = N2, N = N,
                 Group = factor(dataset_jags$Group),
                 Photo = factor(dataset_jags$Photo)
                 )

init <- list(mu = c(7.3, 8.7), sigma.e = 1.7, sigma.P = 0.5)
mcmc <- jags.model("censoredMixedModel.txt",
                   data = win.data,
                   inits = init,
                   n.chains = 4,
                   n.adapt = 100)
params <- c("mu", "sigma.e", "sigma.P",
            "sigma2p", "sigma2e", "diff")
res3 <- coda.samples(mcmc, params, n.iter = 10000)
burn.in <- 1000
```

```
summary(window(res3, start = burn.in))
```

```
##
## Iterations = 1000:10100
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 9101
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD  Naive SE Time-series SE
## diff    -1.4661 0.4388 0.0022999      0.0059050
## mu[1]    7.2331 0.3225 0.0016901      0.0042751
## mu[2]    8.6991 0.3044 0.0015953      0.0043835
## sigma.P  0.7698 0.2723 0.0014271      0.0084074
## sigma.e  6.6044 0.1377 0.0007215      0.0009768
## sigma2e 43.6369 1.8203 0.0095403      0.0129287
## sigma2p  0.6667 0.4593 0.0024070      0.0101675
##
## 2. Quantiles for each variable:
##
##             2.5%     25%     50%     75%   97.5%
```

```
## diff    -2.35715 -1.7512 -1.4558 -1.1757 -0.6203
## mu[1]    6.57784  7.0283  7.2396  7.4462  7.8466
## mu[2]    8.09285  8.5024  8.6991  8.8933  9.3095
## sigma.P  0.22146  0.6003  0.7626  0.9334  1.3252
## sigma.e  6.34142  6.5095  6.6026  6.6963  6.8787
## sigma2e 40.21362 42.3734 43.5942 44.8410 47.3166
## sigma2p  0.04905  0.3604  0.5815  0.8713  1.7562
```

We can see that the the expected difference between the producers is equal to -1.477, and that there is a 95% posterior probability that the true difference is between -2.3763 and -0.6164 - so it is different from 0. (Do note, however, that other runs of the model may bring slightly different results). The random effect for the photos has a variance component of 0.66, and its credible interval does not contain 0.

References

Plummer, M (2016) *Rjags: Bayesian graphical models using mcmc*

Spiegelhalter, D, A Thomas, N Best, D Lunn (2003) *WinBUGS user manual.* version